

Scripting batch

Apprendre à créer des scripts Batch.



Référence : EF-TAI-BATCH

Auteur(s) :
Christophe TABARY

Destinataire(s) :
Easyformer

Date de modification : 21/09/22

Version : 1

Easyformer - 12 rue des violettes - 95000 Cergy
Email : info@easyformer.fr Web : www.easyformer.fr

Sommaire

page



1 Introduction

Batch, qui signifie lots en anglais, est un enchainement automatique de commandes. Il est utilisé pour mettre en place des fichiers script exécutable sur Windows. Généralement ces fichiers ont pour extension .bat ou .cmd et lorsqu'ils sont exécutés ils ouvrent une fenêtre d'invite de commande.

Ils sont simplement composés de commande MS-DOS en clair qui vont s'exécuter les unes après les autres. Chaque commande qui compose un script batch peut être exécutée manuellement dans une invite de commande, à part quelques exceptions qui ne sont utilisable que dans les fichiers script du fait de leur inutilité dans l'environnement DOS (pause, if, for, etc.).

Mais alors c'est quoi le MS-DOS ? MS-DOS (Microsoft Disk Operating System) est tout simplement le système d'exploitation développé par Microsoft pour l'IBM PC dans les années bien avant l'arrivée de Windows. C'est une interface en ligne de commande mono-tâche et mono-utilisateur.

Voici un exemple plus parlant. Imaginons ce script

```
cd \
cd games
Jeux.exe
```

Ce petit script est aussi exécutable manuellement commande par commande. Vous auriez le même résultat en allant dans une invite de commande et en les tapant une par une

```
C:\Users\Martin> cd \ <Entrée>
C:\> cd games <Entrée>
C:\games> Jeux.exe <Entrée>
```

L'intérêt des batch est d'automatiser des tâches. Ils sont très simples à créer car il suffit d'un éditeur de texte pour les écrire. Leur taille est relativement légère et peuvent utiliser toute les commandes DOS.

En revanche c'est un langage non compilé, il est interprété par cmd.exe qui se trouve dans C:\Windows\System32. De ce fait son exécution est plus lente car il n'est pas écrit en langage machine. Ensuite le fichier batch est directement éditable donc non protégé de la copie. Et pour finir certaines opérations élémentaires telles que le traitement de chaines de caractères ou d'opération mathématique n'existent pas sous DOS



2 Quelques exemples de commande MS-DOS

Pour se familiariser un peu avec le DOS voici quelques commandes de base et simple à comprendre.

- **cd :** Affiche le nom du répertoire courant ou change le répertoire courant
- **cls :** Efface le contenu de la fenêtre d'invite de commande
- **copy :** Copie un fichier d'un emplacement à un autre
- **del :** Supprime un ou plusieurs fichiers
- **dir :** Affiche la liste des fichiers et sous-dossiers contenus dans le répertoire.
- **exit :** Quitte le programme cmd.exe (interpréteur de commandes) ou le script batch courant.
- **Type :** Affiche le contenu d'un ou plusieurs fichier texte. Peut aussi en créer un (type nul < test.txt).
- **md :** Crée un répertoire ou un sous-répertoire
- **move :** Déplace un fichier d'un dossier à un autre.
- **path :** Définit le contenu de la variable d'environnement PATH qui contient la liste des répertoires utilisés pour chercher les fichiers exécutables
- **pause :** Commande utilisée pour suspendre l'exécution d'un script batch jusqu'à ce que l'utilisateur presse une touche
- **rd :** Supprime un répertoire
- **ren :** Change le nom d'un fichier ou d'un ensemble de fichiers
- **Xcopy :** Commande puissante avec de nombreuses options pour copier et sauvegarder des fichiers ou des répertoires
- **ipconfig :** Affiche toutes les valeurs de la configuration réseau TCP/IP courante
- **netstat :** Affiche les connexion TCP/IP en cours ainsi que les ports.
- **Nslookup :** permet d'interroger les serveurs DNS pour obtenir les informations définies pour un domaine déterminé (ex : Nslookup google.fr).
- **Tracert :** permet de suivre les chemins qu'un paquet de données (paquet IP) va prendre pour aller de la machine locale à une autre machine connectée au réseau IP
- **Ping :** Envoie une requête ICMP (Internet Control Message Protocol) pour tester la communication entre 2 équipements.
- **Arp :** Affiche et modifie les tables de traduction d'adresses IP en adresses physiques utilisées par le protocole de résolution d'adresses ARP (ex : arp -a).

Bien sûr il en existe de nombreuses autres.



3 Pratiquer le Scripting

3.1 Création d'un fichier Hello

Nous allons créer un fichier qui va afficher à l'écran un simple message Hello world. Ouvrons un éditeur de texte comme le bloc note par exemple. Moi j'utiliserais Notepad++

Commençons par le début. Pour afficher du texte il faut utiliser la commande echo ce qui nous donne ceci

```
@REM désactivation de l'affichage de l'echo
@echo off

echo 3
echo 2
echo 1
echo Hello world

pause
```

Voila à quoi ressemble notre fichier hello.

En premier lieu nous avons utilisé la commande REM qui sert à ajouter un commentaire. On le précède d'un @ pour que celui-ci ne s'affiche pas. L'arobase permet d'afficher uniquement le résulta de la commande sans avoir à afficher la commande elle-même. Sans ce symbole nous verrions s'afficher la commande avant son exécution.

Ensuite nous avons la commande echo off. Un peu comme l'arobase elle va permettre de ne pas afficher les commandes echo mais uniquement leur résultat. On le précède lui aussi d'un arobase pour que la commande ne s'affiche pas.

Les 4 lignes suivantes sont des echo. La commande echo permet l'affichage d'un message. Nous avons donc la commande echo suivi du texte qu'elle doit afficher.

Et pour finir nous avons la commande pause qui permet de mettre le script en pause jusqu'à l'action de l'utilisateur en appuyant sur une touche.

Voici le résultat

```
Invite de commandes - D:\hello.bat

C:\>D:\hello.bat
3
2
1
Hello world
Appuyez sur une touche pour continuer...
```

A savoir que MS-DOS n'est pas sensible à la casse donc majuscule ou minuscule cela n'a pas d'importance.



Maintenant si nous souhaitons enrichir un peu notre fichier et lui donner une meilleure mise en forme. Nous allons ajouter quelques lignes et utiliser une autre commande qui est ECHO.

ECHO. Permet simplement de sauter une ligne. Nous pouvons donc rédiger un petit texte comme ceci

```
@REM désactivation de l'affichage de l'echo
@echo off

echo Bonjour
echo.
echo Je m'apelle Christophe
echo.
echo Bienvenu a tous

pause
```

Voilà ce que ça donne

```
Invite de commandes - D:\hello.bat

C:\>D:\hello.bat
Bonjour

Je m'apelle Christophe

Bienvenu a tous
Appuyez sur une touche pour continuer...
```

Vous constaterez que dans mes textes il n'y à pas d'accent. Batch n'aime pas les accents et si par exemple je mets à accent à bienvenu à tus vola ce que ça va afficher

```
Bienvenu ├á tous
Appuyez sur une touche pour continuer... _
```


3.2 Utilisation des commandes standard dans un fichier batch

Il est très facile d'utiliser des commandes DOS dans un fichier batch, c'est d'ailleurs fait pour ça. Dans notre exemple nous allons nous rendre dans un répertoire lister son contenu et lancer un fichier.

```
@echo off

echo Allons dans le lecteurD
D:

echo allons dans le dossier test
cd test

echo Listons le contenu de ce repertoire
dir

echo executons un fichier
bravo.txt
```

Ici nous utilisons plusieurs commandes pour nous rendre dans le bon répertoire et lancer notre fichier.

D : va simplement nous servir à nous rendre dans le lecteur D

Cd test : Cette commande va nous servir à nous rendre dans le répertoire test qui se trouve à la racine de D

Dir : pour lister le contenu de notre répertoire test

Bravo.txt : va ouvrir notre fichier texte.

Pour info : Pour lancer un fichier .EXE, .BAT ou .CMD il n'est pas nécessaire de préciser l'extension.

3.3 La variable PATH

Cette variable stock les chemins d'accès des commandes. C'est-à-dire que si votre chemin d'accès fait partie du PATH vous pourrez exécuter les commandes qui y sont présente depuis n'importe où dans votre arborescence.



Voici un exemple de PATH

```
C:\>path
PATH=C:\Program Files (x86)\VMware\VMware Workstation\bin\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Wi
ndows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Comm
on;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Program Files\dotnet\;C:\Tools\Phyton\Scripts\;C:\Tools\Phyton\
;C:\Users\ctaba\AppData\Local\Microsoft\WindowsApps;
```

On voit que le premier répertoire de notre PATH est program Files. Cela signifie que je peux exécuter n'importe quel fichier .EXE, .BAT ou .CMD qui se trouvent dans ce répertoire depuis n'importe où.

Path est une variable d'environnement ce qui signifie qu'elle est disponible n'importe où n'importe quand dans le DOS. Elle est modifiable et nous allons voir comment ajouter un chemin d'accès.

Une variable d'environnement est identifiée à la lecture lors d'une définition par deux "%" autour de lui. En effet, à l'exécution, MS-DOS remplace le contenu d'un "mot " entouré par deux "%" par sa valeur. On peut donc lire la variable Path en l'appelant de cette manière %PATH%

Donc pour ajouter un chemin à notre variable nous allons utiliser la commande

```
PATH=%PATH% ;CHEMIN_A_AJOUTER
```

Le premier Path est en écriture donc pas besoin de l'entourer du %. En revanche le second PATH est destiné à inclure l'ancien PATH et doit contenir la variable PATH d'où la présence des %. Le point-virgule sert à séparer la variable du chemin d'accès à ajouter.

Si par exemple je souhaite inclure le chemin D:/test dans mon PATH j'utiliserais la commande

```
PATH=%PATH% ;D:/test
```

3.4 Paramètres envoyés à une commande ou à un programme

On appelle paramètre tous les arguments passés à un programme ou une commande. Les paramètres sont séparés par des espaces.

Par exemple : Format E : /V :nom_du_volume /B /C

Format est une commande DOS. Cette commande reçoit donc comme argument tous les paramètres envoyés par l'intermédiaire du DOS.

Il est possible de lire les différentes variables relatives aux arguments passé à votre programme. Ces variables vont de %0 à %9 . La variable %0 contient le chemin d'accès au programme, %1 le premier paramètre, %2 le second paramètre… jusqu'à %9 qui contient le neuvième paramètre envoyé au batch.



Prenons l'exemple de notre fichier hello.bat éditons le et mettons ça à l'intérieur

```
@echo off
echo L'adresse de ce fichier est %0
echo Le premier parametre est %1
echo Le second parametre est %2
echo Le troisieme parametre est %3
echo Le quatrieme parametre est %4
```

Maintenant si nous l'exécutons voila ce que nous obtenons

```
C:\Users\ctaba>D:\hello.bat
L'adresse de ce fichier est D:\hello.bat
Le premier parametre est
Le second parametre est
Le troisieme parametre est
Le quatrieme parametre est
```

Nous avons bien en variable %0 avec l'adresse du fichier hello, et pour les autres variables elles sont vides car nous n'avons pas ajouté de paramètre à l'exécution e notre fichier.

Donc si nous rajoutons plusieurs paramètres factices pour tester nous auront ceci

```
C:\Users\ctaba>D:\hello.bat /C /E /V
L'adresse de ce fichier est D:\hello.bat
Le premier parametre est /C
Le second parametre est /E
Le troisieme parametre est /V
Le quatrieme parametre est
```

Ici on voit bien que les 3 paramètres ajouté au lancement de notre fichier ont bien été prise en compte.

3.5 Variables d'environnement

Une variable d'environnement représente une valeur accessible n'importe où et n'importe quand dans l'environnement DOS. Pour visualiser les variables d'environnement actives sur votre ordinateur, il vous suffit de taper la commande set ce qui donne par exemple :

```
TEMP=C:\Users\ctaba\AppData\Local\Temp
TMP=C:\Users\ctaba\AppData\Local\Temp
USERDOMAIN=CHRISP
USERDOMAIN_ROAMINGPROFILE=CHRISP
USERNAME=ctaba
USERPROFILE=C:\Users\ctaba
windir=C:\Windows
ZES_ENABLE_SYSMAN=1
```





Précisons que, dans le langage Batch, la seule façon de stocker des données est de les associer à des variables d'environnement. Il n'existe pas de variables "locales" que d'autres fichiers Batch ne pourraient pas connaître.

Pour définir une variable d'environnement il suffit de taper la commande

Set Nom_Variable = Valeur de la variable

Si par exemple nous voulons définir une variable contenant la version de Windows il suffit de taper Set VersionWindows = 11

A la validation de la commande rien ne s'affichera car aucun résultat d'affichage n'est demandé mais si nous retapons notre commande set nous pourrons voir notre nouvelle variable

```
VersionWindows = 11
windir=C:\Windows
ZES_ENABLE_SYSMAN=1
```

En cas d'erreur inutile d'effacer la variable pour la recréer, il suffit de la redéclarer avec la nouvelle valeur et l'ancienne sera écrasé.

Pour la supprimer il suffit de lui attribuer une valeur vide.

Pour utiliser une variable d'environnement, il faut l'encadrer par des "%". Il n'y a que la commande SET qui ne demande pas de signe "%" pour l'argument représentant le nom de la variable à définir : en effet, on ne lit pas la valeur de la variable, mais on la définit. Lors de l'exécution du Batch, lorsque l'interpréteur rencontre un nom encadré de deux "%", il substitue ce nom par la valeur de la variable qu'il représente, si elle existe.

Voici un petit exemple de script pour lire une variable et la remplacer

```
1 @echo off
2 set VersionWindows=11
3 echo %VersionWindows%
4 echo La version de Windows est %VersionWindows%
5 Set VersionWindows=%VersionWindows%-64bits
6 echo %VersionWindows%
```

La première ligne nous la connaissons je ne vais pas l'expliquer à nouveau. La seconde ligne va définir la valeur de notre variable. En ligne 3 nous allons juste l'afficher. En ligne 4 on l'affiche toujours mais avec une phrase qui la précède. En ligne 5 Nous lui donnons une nouvelle valeur. Et pour finir en ligne 6 on la réaffiche. Voici ce que ça donne en invite de commande

```
D:\>hello
11
La version de Windows est 11
11-64bits
```



Il est aussi important de savoir que l'invite de commande fonctionne en instance avec un système d'héritage. C'est à dire que si je définis des variables dans une fenêtre d'invite de commande elles n'existeront pas dans une autre invite de commande.

```
Invite de commandes

D:\>set VersionWindows
VersionWindows=11

D:\>
```

```
Invite de commandes

D:\>set VersionWindows
La variable d'environnement VersionWindows n'est pas définie.

D:\>
```

Mais si en revanche j'ouvre une autre instance à partir de ma fenêtre principale en lançant un cmd dans ma fenêtre existante elle va hériter de la première fenêtre et ma variable sera toujours connu

```
D:\>set VersionWindows
VersionWindows=11

D:\>cmd
Microsoft Windows [version 10.0.22000.978]
(c) Microsoft Corporation. Tous droits réservés.

D:\>set VersionWindows
VersionWindows=11
```

Maintenant si je modifie ma variable dans ma nouvelle instance et que je la quitte pour revenir sur mon instance principale la modification ne sera pas prise compte. L'héritage n'est pas à double sens.

```
D:\>set VersionWindows
VersionWindows=11

D:\>cmd
Microsoft Windows [version 10.0.22000.978]
(c) Microsoft Corporation. Tous droits réservés.

D:\>set VersionWindows
VersionWindows=11

D:\>set VersionWindows=XP

D:\>set VersionWindows
VersionWindows=XP

D:\>exit

D:\>set VersionWindows
VersionWindows=11
```




3.6 Saut inconditionnel

Le langage Batch vous permet d'utiliser des commandes de boucle, c'est à dire de répéter un bloc de commandes indéfiniment. Nous allons étudier dans ce chapitre la commande "Goto". C'est une commande de saut inconditionnelle, qui ne peut être arrêtée (ou à l'aide de commandes spécifique que nous n'avons pas encore vu), par conséquent vous allez être amené à fermer de façon "brutale" des programmes DOS.

En principe, les lignes de commande sont traitées les unes après les autres dans un fichier Batch. Toutefois, dans certains cas, on est obligé de sauter des lignes pour reprendre le traitement à un autre endroit du fichier. C'est dans ces cas-là que nous allons utiliser les commandes de boucle.

On associe souvent une commande de saut à une commande d'instruction conditionnelle (que nous verrons dans le chapitre suivant), ou lorsqu'un bloc de commande doit être répété indéfiniment. C'est sur ce cas que nous allons nous pencher pour l'instant.

Pour faire une boucle, il nous faut deux commandes

- La première est un "Label", c'est-à-dire une étiquette posée dans le programme à l'endroit où la boucle doit commencer. Une sorte de marqueur qui sert de point de repère.
- La seconde est la commande Goto, (de l'anglais Go To… qui signifie "aller à") qui, accompagnée du nom du Label, indique à l'ordinateur, quand il doit se rendre à l'étiquette du même nom. Par exemple :

Commande 1
Commande 2
Label BONJOUR
Commande 3
Commande 4
Commande 5
Goto BONJOUR

Les commandes 1, et 2, sont exécutées une fois, alors que les autres commandes sont exécutées en boucle, puisque le programme rencontre "GOTO", va au label du même nom, continue, rencontre à nouveau "Goto", retourne au label , etc…

Un label se présente sous la forme *:NomDuLabel*

Le nom ne doit pas dépasser 8 lettres (si le nom du label dépasse 8 lettres, seules les 8 premières lettres seront prises en compte), et ne pas être composé d'espaces. Par exemple

:Debut



Un "Goto" se présente sous la forme de cette commande suivie du nom du label, par exemple

Goto Debut

Nous allons maintenant créer une boucle infinie. Pour cela utilisons notre premier script pour afficher Hello World et affichons-le indéfiniment.

```
@echo off
:Debut
echo Hello, Word !
Goto Debut
```

Et voila le résultat

```
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
```

Et le script continue de se dérouler tant qu’on n’y met pas fin. Pour y mettre fin il suffit de faire la combinaison CTRL C, On va nous demander confirmation pour arrêter le script et voilà le script s’arrête.

```
Hello, Word !
Hello, Word !
Hello, Word !
Hello, Word !
^CTerminer le programme de commandes (O/N) ?
```

3.7 L’instruction conditionnel, la commande IF

Voici une commande qui permet d'introduire des conditions dans les fichiers batch. Si la condition formulée est remplie, le reste de la ligne de commande est exécutée, et le programme continue normalement, sinon le reste de la ligne n'est pas exécuté, et le programme continue également.

Attention : seule la fin de la ligne est exécutée, par conséquent seule 1 seule commande peut être conditionnelle, ce qui peut parfois poser des problèmes. Dans ce cas, utilisez la commande GOTO pour aller à un endroit particulier si la condition est remplie.



Syntaxe d'utilisation :

```
If "<Condition>"=="<Valeur>" <Action>
```

Attention il est important de :

- Toujours encadrer la condition et la valeur à tester par des guillemets
- De veiller à utiliser, lors d'un test, le double signe égal (== au lieu de =) Le signe égal permet d'assigner une valeur alors que le double égal est là pour effectuer une comparaison. ATTENTION : La comparaison et sensible à la casse, il faut donc faire attention aux majuscules. Dans notre cas /AIDE sera donc différent de /Aide
- Se rappeler que "<Action>" ne représente qu'une seule commande à exécuter.

Mais alors pourquoi faut-il absolument encadré ces valeurs par des guillemets ? Parce qu'à l'exécution, la valeur des variables vient remplacer leur écriture, et si une variable est nulle, MS-DOS génère une erreur car il ne peut comparer un terme qui n'existe pas. Par compte, s'il y a des guillemets, MS-DOS "comprend" qu'il fait une comparaison avec une variable vide.

Exemple :

```
If "%1"=="/AIDE" echo Ce texte s'affichera
```

```
@echo off
If "%1"=="/AIDE" echo Ce texte s'affichera
```

```
D:\>hello /AIDE
Ce texte s'affichera
```

Ici, on va être amené à comparer le contenu de la variable d'environnement paramètre n°1 avec le texte "/AIDE". Si ceux-ci sont identiques, un texte sera affiché à l'écran.

Il existe d'autre variante de la condition IF qui permettent une plus grande flexibilité

- **IF NOT (condition)**

Vérifie si la condition est remplie. Si oui, la ligne suivante est traitée, sinon, le reste de la commande est exécutée. C'est en fait "l'inverse" de la commande IF.

```
If not "%1"=="/AIDE" echo Ce texte s'affichera
```

```
@echo off
If not "%1"=="/AIDE" echo Ce texte s'affichera
```

```
D:\>hello /AIDE

D:\>
```



- **IF EXIST (fichier)**

Vérifie l'existence du fichier désigné. S'il existe, le reste de la ligne est traité, sinon on passe à la ligne suivante. Ce type de commande peut-être aussi utilisé sous la forme "If not exist", dans ce cas le reste de la commande est traité que si le fichier n'existe pas. Il est aussi important de noter que vous n'êtes pas obligé d'utiliser des guillemets puisque le paramètre représentant le fichier ne peut-être nul.

```
If exist d:\hello.bat Copy hello.bat hello.old
```

```
@echo off
If exist d:\hello.bat Copy hello.bat hello.old

```

```
D:\>hello
        1 fichier(s) copié(s).
```

On peut utiliser une condition IF couplé à une boucle goto qui va permettre d'exécuter différentes commandes selon le résultat de la condition. Nous avons utilisé la commande IF pour introduire des questions dans les fichiers Batch. Il serait souhaitable maintenant d'utiliser plusieurs commandes en fonction du résultat de la question. Par exemple

```
1 @echo off
2 set V=toto
3 if "%V%"=="toto" goto suite
4 echo V n'est pas egal a toto
5 :suite
6 echo V est egal a toto
```

Ici nous allons définir la variable V en lui donnant la valeur toto. Ensuite nous allons développer une condition qui dit que si notre variable V est égal à toto alors il doit se rendre au label suite. Sinon il exécutera la ligne 4. Bien sur ici c'est une évidence puisque nous avons définit la variable au préalable mais c'est juste un exemple.

Un autre exemple. Que se passera t'il dans ce cas de figure là ?

```
1 @echo off
2 If not "%1"=="/?" Goto Suite
3 Echo Voici l'aide de ce programme
4 Echo Bla bla bla bla
5 :Suite
6 Echo Pour commencer, pressez une touche
7 Pause
```

D'après ce que nous savons la ligne 2 nous indique que si le premier paramètre n'est pas égal à "/?", alors il doit se rendre au label suite, sinon il continue en ligne 3. Mais là il y a un problème. Imaginons que la condition ne soit pas remplie. Le programme continue de



s'exécuter et passe donc en ligne 3 ensuite en ligne 4. Mais va-t-il s'arrêter là ? Et bien non, il va continuer ligne 5, 6 et 7. Pour éviter cela 2 solutions. La première que l'on connait déjà serait de rajouter un goto avant le label suite qui redirigerais vers un nouveau label en fin de script. Ce qui nous donnerais ceci

```
@echo off
If not "%1"=="/?" Goto Suite
Echo Voici l'aide de ce programme
Echo Bla bla bla bla
goto fin
:Suite
Echo Pour commencer, pressez une touche
Pause
:fin
```

Cela fonctionne mais nous avons mieux. Nous avons la commande exit. Utilisé tel quel, elle met fin au CMD et fermera donc notre invite de commande. Pour éviter cela et mettre uniquement fin au script nous allons lui ajouter le paramètre "/B". Ce qui nous donne donc ceci

```
@echo off
If not "%1"=="/?" Goto Suite
Echo Voici l'aide de ce programme
Echo Bla bla bla bla
exit /B
:Suite
Echo Pour commencer, pressez une touche
Pause
```

Ainsi notre script est plus clair.

Vous pouvez aussi utiliser la commande goto :eof. Cette commande sert tout simplement à sortir du script. A savoir que :eof n'est pas un label qu'il faut positionner dans le script.

```
@echo off
if not "%1"=="/?" goto suite
echo voici l'aide de ce programme
echo blablabla
goto :eof
:suite
echo pour commencer, pressez une touche
pause
```




3.8 Les boucles

Une boucle, comme son nom l'indique, est un processus qui va se répéter un certain nombre de fois. Nous avons vu qu'avec une condition IF et un goto nous pouvions effectuer une boucle mais là nous allons faire un peu mieux et utiliser des éléments plus pertinents pour une meilleur flexibilité de notre boucle. Prenons l'exemple suivant

```
@echo off
for %%A in (1 2 3 4) Do Echo C'est le nombre %%A
```

Ceci est une boucle FOR...DO. Elle va permettre d'effectuer plusieurs fois un afficha echo. Tout d'abord nous constatons que notre variable A est précédé de 2 % alors que nous n'en avons toujours mis qu'un seul. Une variable avec %% est une variable spécial créé pour la boucle for pour représenter l'élément de boucle courant. Cette variable prend alors toutes les valeurs de la liste spécifiée entre les parenthèses : dans notre cas, %%A prend donc successivement les valeurs 1, 2, 3, et 4. Les valeurs constituant la liste doivent être séparées entre elles par des espaces, des virgules, ou des points-virgules. Ensuite, la commande qui suit immédiatement est exécutée avec la valeur prise par la variable %%A. Dans notre cas, on verra à l'écran le message "C'est le nombre" suivi de la valeur de la variable à chaque exécution de ECHO.

Un autre intérêt de cette commande est que les éléments de la liste peuvent-être des noms de fichiers. Ainsi il est possible d'exécuter une seule commande pour plusieurs fichiers. Vous pouvez donc afficher à l'écran plusieurs fichiers à la fois avec une seule commande qui est TYPE. Par exemple

```
@echo off
FOR %%A IN (test.txt toto.txt) DO TYPE %%A
```

Cette boucle va afficher le contenu des 2 fichiers texte.

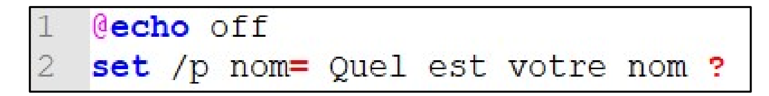
3.9 Interagir avec l'utilisateur

Il est possible d'interagir avec l'utilisateur et lui demander de saisir des données. Pour cela nous allons utiliser la commande set /p.

```
Set /p variable=[chaine_de_caractère]
```

"Variable" correspond au nom de la variable qui contiendra les données saisies par l'utilisateur. Et "chaine_de_caractère" correspond au message qui s'affichera à l'utilisateur.

Voici un exemple qui demande son nom à l'utilisateur.



```
@echo off
set /p nom= Quel est votre nom ?
```

3.10 Ecrire dans un fichier

Vous pouvez écrire dans des fichiers, à l'aide de commande Batch. Nous avons dit dans le chapitre 1 que la commande ECHO servait en fait à "écrire" quelque chose quelque part. Pour l'instant, nous nous sommes contentés "d'écrire" sur l'écran, mais rien ne nous empêche de le faire sur le disque. Nous allons aussi utiliser les chevrons (">" ou "<") comme caractères de redirection. Vous devez veiller au nombre de chevrons, et à leur sens, en effet, la sortie sur le fichier en dépendra.

3.10.1 Ecriture en mode Ajout

Ce mode permet d'ajouter des données sans écraser celles qui étaient inscrites précédemment dans le fichier. Nous allons utiliser 2 chevrons, orientés vers la droite, qui pointent vers le nom de fichier à utiliser

```
@echo off
Echo bonjour a tous>>D:\texte.txt
```

Ainsi, tout le texte compris entre "Echo" et les ">>" sera écrit dans "d:\texte.txt". Si le fichier n'existe pas, il sera créé et les données y seront inscrites sans générer d'interruptions ou d'erreurs, sauf si le ou les répertoires le contenant n'existent eux même pas. Le texte à inscrire sera ajouté à la fin du fichier Une nouvelle ligne sera créée dans le fichier à chaque fois que vous appellerez la commande.

3.10.2 Ecriture en mode écrasement

Contrairement au mode d'ajout, le mode d'écrasement efface toutes les données inscrites précédemment dans le fichier, puis inscrire la ligne transmise. Nous allons utiliser 1 seul chevron, orienté vers la droite, qui pointe vers le nom de fichier à utiliser.

```
@echo off
Echo bonjour a tous>D:\texte.txt
```

Le contenu du fichier sera automatiquement effacé**.** Toutes les données seront perdues et remplacées.

3.10.3 Ecrire le résultat d'une commande dans un fichier

Vous pouvez inscrire le résultat d'une commande DOS dans un fichier, avec les deux modes décrits plus haut ("Écrasement" et "Ajout"). Pour cela, vous n'avez qu'à supprimer "Echo", et remplacer le texte à écrire dans le fichier par une commande MS-DOS.

```
dir c:\*.*>>c:\listing.txt

```

Le contenu du disque C:\ sera inscrit en mode "rajout" dans le fichier listing.txt



3.10.4 La redirection vers "nul"

"Nul" représente un périphérique virtuel inexistant. Utilisé avec ">" et ">>", il permet "d'écrire" le résultat de commande vers rien du tout, c'est à dire, en clair, de les masquer. Par exemple lorsque l'on utilise la commande pause nous avons un texte qui s'affiche à l'écran (Pressez une touche pour continuer). Et bien pour masquer ce texte tout en conservant la commande

```
@echo off
pause>Nul
```

3.11 Appel d'autres fichiers Batch

La commande CALL permet d'appeler un fichier Batch à partir d'un autre fichier batch. Après avoir traité le fichier batch appelé, le programme revient au premier fichier batch et à l'endroit précis où le fichier batch a été appelé. Vous pouvez également appeler un fichier batch à partir d'un autre sans pour autant revenir au fichier batch de départ. Il suffit tout simplement d'appeler le fichier batch par son nom (ou son adresse) c'est à dire sans CALL.

- **Appel sans call :**

Vous pouvez appeler un fichier batch à partir d'un autre en utilisant son nom. Le résultat est que le batch appelé est traité, mais il est impossible de revenir au batch de sortie précédemment traité. On peut en quelque sorte parler de "liaison unilatérale".

```
@echo off
C:\MesBatch\fichier.bat
Commande 2
Commande 3
Commande 4
```

Dans cet exemple les commandes 2, 3 et 4 ne s'exécuteront pas car une fois le fichier.bat appelé la sortie du bat d'origine est définitive.

- **Appel avec call :**

Un batch X appelle un batch A à un endroit précis. CALL a pour rôle de contrôler que MS-DOS remarque bien le "point de saut" et revienne dans le batch appelant après avoir traité le batch appelé. Le Batch A est donc utilisé comme un sous-programme. Cette utilisation comporte un avantage majeur : on doit programmer une seule fois les routines batch et on peut ensuite les appeler le nombre de fois que l'on veut à partir de n'importe fichier Batch.



```
@echo off
call C:\MesBatch\Routine.bat
Commande 2
Commande 3
Commande 4
```

Ici le fichier Routine.bat est appelé à l'intérieur du bat d'origine donc une fois le script de routine terminé le script reprend là où il en était.

3.12 Les opérateurs

Nous avons vu précédemment que nous pouvions effectuer des comparaisons avec le symbole ==. Mais celui est limité, il ne peut renvoyer que 2 réponses. Il existe donc d'autres opérateur de comparaisons qui, dans une condition IF, peuvent permettre plus de résultats. En voici la liste

Opérateur	Signification
EQU	Egal à
NEQ	Différent de
LSS	Strictement plus petit que
LEQ	Inférieur ou égal à
GTR	Strictement plus grand que
GEQ	Supérieur ou égal à

Voici un exemple

```
@echo off
set A=1
set B=2

if %A% LSS %B% goto :fin
echo B est finalement plus petit que A
exit /B
:fin
echo B est bien plus grand que A
```

Ici j'ai 2 variables A et B. A est strictement inférieur à B. Je lui dis que si A est strictement plus petit que B alors il doit se rendre au label fin, sinon il continue.



Il existe aussi des opérateurs arithmétiques que voici

Symbole	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division

4 Quelques éléments utiles

Voici quelques éléments pratiques qui viennent alimenter tout ce que l'on a déjà vu précédemment.

- **Set /a :**

La ligne de commande n'est pas prévue pour gérer les fonctions mathématiques mais il est possible d'exécuter des opérations très simples d'entiers avec les variables. Un commutateur "/a" a été ajouté à la commande "set" pour permettre quelques fonctions basiques. Cela est principalement utilisé pour effectuer des additions ou des soustractions. Par exemple, il est possible d'incrémenter ou de décrémenter un compteur dans une boucle. En principe, il est également possible d'effectuer des multiplications et des divisions, mais seuls quelques nombres peuvent être gérés donc l'utilisation en pratique est limitée. Bien que les variables soient stockées comme des chaînes, l'interpréteur de commande reconnaît les chaînes qui ne contiennent que des entiers, et leur permet d'être utilisées dans des expressions arithmétiques.

```
@echo off
set /a compteur=%compteur%+1
```

- **Nombre aléatoire :**

Pour obtenir un nombre aléatoire il faut utiliser la commande %RANDOM%. Random va vous donner un nombre aléatoire compris entre 0 et 32767, mais vous pouvez personnaliser cet intervalle. Voici quelques exemples

Ici je vais simplement laisser l'intervalle d'origine et afficher le nombre à l'écran.

```
@echo off
set Num=%Random%
echo Votre nombre aleatoire est %Num%
```




Maintenant si je veux un nombre aléatoire dans un intervalle de 10 à 20 je vais écrire ceci

```
@echo off
set /a Num=%RANDOM% * (20 - 10 +1) /32768 + 10
echo %Num%
```

Pour définir l'intervalle entre parenthèse je commence par son maximum et ensuite son minimum.

- **Condition if :**

Nous avons déjà vu la condition if mais sachez qu'elle existe aussi sous d'autre forme que celle que nous avons utilisé. Voici quelques commande if possible

1) If else

Permet d'effectuer 2 actions différente selon le résultat.

ATTENTION : les parenthèses ne sont pas à mettre n'importe où. Si par exemple je retire ma parenthèse ouverte qui est en fin de ligne 4 et que je la déplace en début de ligne 5, ça peut paraitre identique mais cela ne fonctionnera pas.

```
1 @echo off
2
3 set A=2
4 if %A% EQU 2 (
5 echo A est egal a 2
6 ) else (
7 echo A n est pas egal a 2
8 )
```

2) If exist

Permet de tester l'existence d'un fichier et d'effectuer des commandes en fonction du résultat.

```
@echo off
if exist hello.bat (
echo mon fichier hello existe
) else (
echo mon fichier hello n existe pas
)
```



On peut aussi l'utiliser à la forme négative

```
@echo off
if not exist hello.bat (
echo mon fichier hello n existe pas
) else (
echo mon fichier hello existe
)
```

3) If defined

Cette condition permet de tester l'existence d'une variable.

```
@echo off

set A=2
if defined A  (
echo A existe
) else (
echo A n existe pas
)
```

